acorn electron

ME & MY MICRO

ME & MY MICRO

Contents

Acknowledgements to Richard Freeman

First published in 1984 by Acornsoft Limited

FIRST EDITION

Acornsoft Limited, Betjeman House, 104 Hills Road,
Cambridge CB2 1LQ, England. Telephone (0223) 316039

INTRODUCTION

The eight games included in this package are:

MONSTERZAP	ANAGRAMS
BOMBER	TWISTER
MAZERACE	MATCHEM
KONGO	PAIRS

The games on this cassette are extensions to the 'core' programs featured in the Yorkshire Television series, 'Me and My Micro'. They have been designed to encourage you, the player or programmer, to find out more about BASIC programming.

These games are much slower and less dramatic than most arcade games. But they are all written in BASIC, and clearly laid out, using labelled subroutines, so that you can follow exactly how they work.

To find out what is going on you should break into the programs and look at the listings. Then you can alter them, slow them down, speed them up, and if you find a more elegant way of doing something, or an improvement, then produce your own version. These programs are the starting point from which you can get to grips with your micro, and make it work for you.

Loading instructions

To load and run the first program place the cassette (fully rewound) in the cassette recorder, type

```
CHAIN "MONSTERZAP"
```

and press RETURN; the 'Searching' message should appear on the screen as you do this. Now press the PLAY button on the cassette recorder and wait approximately one minute for the program to load. The game will start as soon as loading is complete.

The other programs can be loaded in the same way using the CHAIN command.

If you don't want to run the program then simply LOAD it off the cassette rather than CHAINing it, for example type

```
LOAD "TWISTER"
```

and press RETURN; the 'Searching' message should appear as you do this. Now press the PLAY button on the cassette recorder and wait for the program to load.

Breaking into the programs

If you want to break into the program while it is running then answer N to the question

```
Do you want another game (Y/N)?
```

This will set MODE 6 and return to BASIC, ready for you to LIST. Then you can go to work and change things if you want. If you don't want to wait until the end of the program then press ESCAPE, this will cause the game to stop at that point. You can then clear the screen and LIST as above.

PLAYING INSTRUCTIONS

To play any of the games refer to the following instructions:

MONSTERZAP
The Microids have invaded our streets. Use the F key to fire your laser. You have to zap the monsters within your 20 shots, without hitting the buildings that they sit between.

BOMBER
Your aeroplane is gliding downwards over a deserted city. Use the F key to level the skyscrapers to the ground before you crash into them. You have only 20 shots to ensure a safe landing for your aeroplane. Good luck.

MAZERACE
Manoeuvre the Quacman through the maze in the shortest time possible. Watch out for the delaying puddles of glue. Use these controls:

[*] Up

[/] Down

[X] Right

KONGO
Kong the gorilla is leaping towards the damsel to carry her off. You must collect a basket at each level, in order to climb the ladder and rescue the damsel from Kong. Use these controls:

[*] Up

[X] Right

[Z] Left

ANAGRAMS
The first player enters a word, the computer rearranges it, and the second player must guess that word, letter by letter.

TWISTER
Rearrange the jumbled numbers into the correct sequence in the fewest number of twists.

MATCHEM
Remember where particular cards are, and try to find all the matching pairs in the least number of tries. Alternatively try to match more than your opponent.

PAIRS
There are only three pairs. To pick a card simply press the right number key. Having mastered PAIRS the younger members of the family might like to attempt MATCHEM.

Modifying the programs for the BBC Microcomputer

Most of the games will have to be slowed down. You can do this by FOR – NEXT loops just to hold up the program at some point, or you could add more sounds, colour, movement and so on.

Getting the programs to run, load, and breaking in are all exactly the same as on the Electron.

PROGRAM OUTLINES

The arrangement and theory of each game is explained here along with certain points specific to the Electron. By looking at both the structure of each program, and the listings, you should be able to see how ideas can be broken down, detail by detail, and turned into BASIC.

All the programs use PRINT TAB(column,row); when printing such things as gorillas, aeroplanes, and anagrams on the screen. In the action games look for the variables 'c' and 'r' – column and row. (Most variable names bear some relation to their use in the program). Other more advanced methods of printing characters on the screen have been deliberately ignored in order that the games remain simple, standard, and easy to understand.

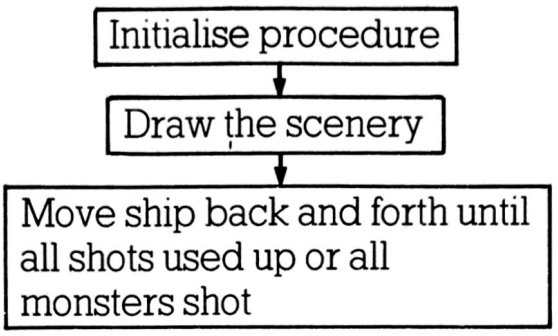
In these programs the standard method of moving an object, or character, has been to delete the object, change its coordinates, and reprint it at the new position. Again more advanced methods have been ignored.

Further information, or clarification of commands you do not understand (in particular the use of Envelopes of *FX calls) can be obtained by referring to the User guide for your machine.

MONSTERZAP

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Set up user-defined character definitions.
Set variables to initial values.
Set up envelopes.

Draw the scenery

Print sky and ground.
Plot random stars.
Print skyscrapers and monsters.

Movement procedure

Print the ship and then wait for a while so it can be seen well.
Look at the keyboard to see if the F (fire) key is being pressed, if it is then carry out the firing procedure.
Print a space at the ship's present position to delete it.
Change the ship's coordinates by one column.

Firing procedure

Make a firing sound.
Print a zap effect on the screen.
Print the rubble left over.
Mark which monster is hit, if all the monsters are gone then finish.
Add one to the shots count.
Display the amount of shots used.
If all the shots are used then finish.

Finish

Print the score and ask if another game is required.

Notes

The following notes contain comments on some of the BASIC used in the program.

Ideally the MODE statement will be in the initialise procedure, but MODE doesn't work in procedures.

VDU 23,1,0;0;0;0; is a method of turning the cursor off to improve the appearance of the game. Changing MODE will bring the cursor back again.

*FX11,0 is an FX call which stops the auto repeat of a key being pressed down. This means the shots can be more precise as you won't fire off 5 at once.

*FX15 (or*FX15,0) is a FX call which flushes (empties) the keyboard buffer. Here, the program prevents you typing several shots ahead, as it clears the stored keypresses away each time it loops round.

*FX12 is a FX call to reset the auto-repeat delay back to their default values so you can type normally again.

If the column number is MOD with 7, (which gives the remainder after division by 7, ie numbers in the range 0 to 6) then the monsters lie from 4 to 6.

So the part of a line

IF(c MOD 7)>3 AND (c MOD 7)<=6

checks if it is a monster that has been hit

AND hit(c)=0

checks it hasn't been hit already

THEN . . . . . :hit(c)=1

marks this spot down as hit.

When the array hit(x) is full of 1's, then all the monsters have been hit.

SOUND &11,1,200,4 uses the ENVELOPE number one, and interrupts any sound that was previously being made. (The &11 part means use channel 1 and interrupt it.)

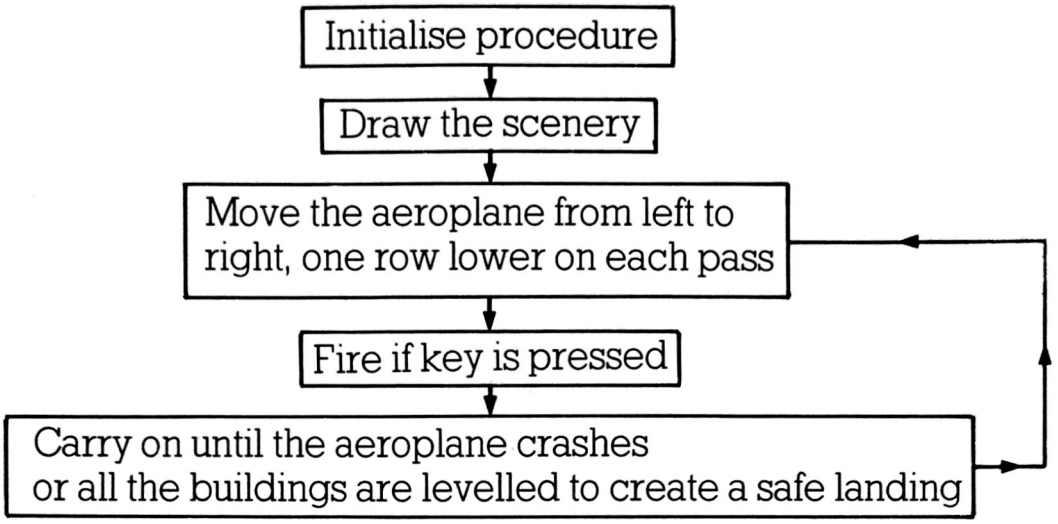
An idea for extending the program

Mark down which buildings are hit, and print a suitably scathing message if all the buildings are destroyed.

BOMBER

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Similarly to Monsterzap, set up the envelopes, user-defined characters, variables and palette.

Draw the scenery

Print sky and ground.
Print the two star characters at random positions.
Print some sets of skyscrapers.

Main movement loop

Move the aeroplane's column position from left to right in a loop.
If a building is detected at the aeroplane's position then use the crash routine.
Print the aeroplane.
If the aeroplane's row is equal to the ground then use the successful landing routine.
If the F key is being pressed then carry out the firing procedure.
Print the character that was there before the aeroplane (ie blank the aeroplane out).
Increase the aeroplane's column by one.

Loop back until the right hand side of the screen is reached.
Bring the aeroplane down by one row (increase the row position value by one).
Loop back until you crash into a building, or a safe landing achieved.

Firing procedure

Make a firing sound.
Wipe out the building in the aeroplane's present column (if there is one there).
Add one to the shots counts.
Display the amount of shots used.
If all the shots have been used then print goodbye.

Crash

Flash the background different colours.
Print the crashed message.
Ask if another game is wanted.

Success

Print the success message.
Make a successful sound.
Ask if another game is wanted.

Notes

The following notes contain comments on some of the BASIC used in the program.

If a building is hit it is marked down in an array, in a very similar way to Monsterzap.

VDU 23;10,32;0;0;0; is another way to turn the cursor off. This is the command to use if your cursor is to disappear on the Electron and all models of the BBC microcomputer.

VDU 19 is used to change the colours on the screen. In MODE 1 only 4 colours are allowed (including background), but these can be selected from any in the palette. Refer to page 102 in the *Acorn Electron User Guide* or page 162 in the *BBC Microcomputer System User Guide.*

Operating system routines may be called from machine code programs and values may be passed to them or passed back by them in the X and Y registers. The normal way of accessing operating system calls from BASIC

is to use a *FX call. Parameters may be passed to the operating system in this way but no values can be returned. The LOOK function has been defined to allow values to be returned. The reason it is used here is to keep the program using PRINT and TAB positions, instead of using graphics coordinates and PLOTing, or using VDU 5. The *FX call used is *FX135, which returns the character at the text cursor in the X register and the graphics mode in the Y register. The function only extracts the X register value to read what character is at a certain TAB position so that stars can be replaced after the aeroplane has moved over them, and the aeroplane crashing into a building can be detected.

BBC users are referred to page 432 of their User Guide where there is a similar routine.

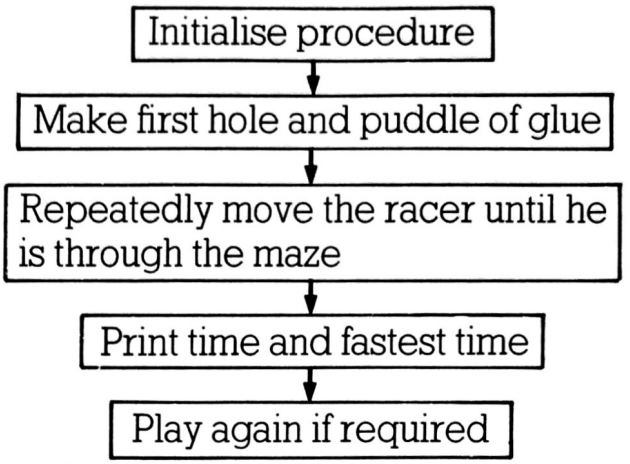
An idea for extending the program

Try making the shot fall as the aeroplane is moving instead of stopping the aeroplane while the shot is being fired, also try and demolish the object that the shot hits first. (Rather than demolishing the object where the shot hits the ground.)

MAZERACE

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Set up envelopes for sound effects.
Set all the variables to their initial values – the racer's starting row and column and the column the first glue puddle appears in.
Draw the maze walls.
Select the required colours from the palette.
Wait for a key to be pressed to start.
Set up user defined characters.

Make a hole/two holes and plant a glue puddle

If the racer is just about to finish the maze then don't make another hole.
Pick two rows at random for the two possible holes to appear in, and if only on hole is wanted then make the two row numbers the same.
Print the two holes.
If a puddle of glue is needed then print it, and set a flag to show it hasn't been stepped in yet.

Move procedure

Print racer.
Make a sound.
Look at which key is being pressed.
Print the time taken so far.
Delete the racer.
If the X key is pressed and the racer is opposite a hole then move right by two columns, and draw the next hole/puddle.
If the / key is pressed then move down by a row.
If the * key is pressed then move up by a row.
If the row coordinates are too large or too small then bring them back into the maze.

Stick in glue

Make a sound (using an envelope).
Cancel the possibility of getting stuck again in this puddle by setting a flag to show it has been stepped in.
Wait until some time has gone by before continuing.

Notes

The following notes contain comments on some of the BASIC used in the program.

*FX11,17 is a *FX call to reduce the delay before a key starts repeating when held down. This makes the game easier to play as the racer moves off instantly at its steady pace (it can make typing difficult if you leave your fingers on the keys).

*FX15 (which is the same as *FX15,0) is a *FX call to flush all the buffers (short term memories). The particular buffer to be emptied in this case is the keyboard buffer. While the racer is stuck in the glue, the computer stores away all the key presses in that time, and then uses them when the delaying routine is finished. Flushing (emptying) the buffer means that you start afresh (in terms of key pressing) when you are freed from the glue puddle.

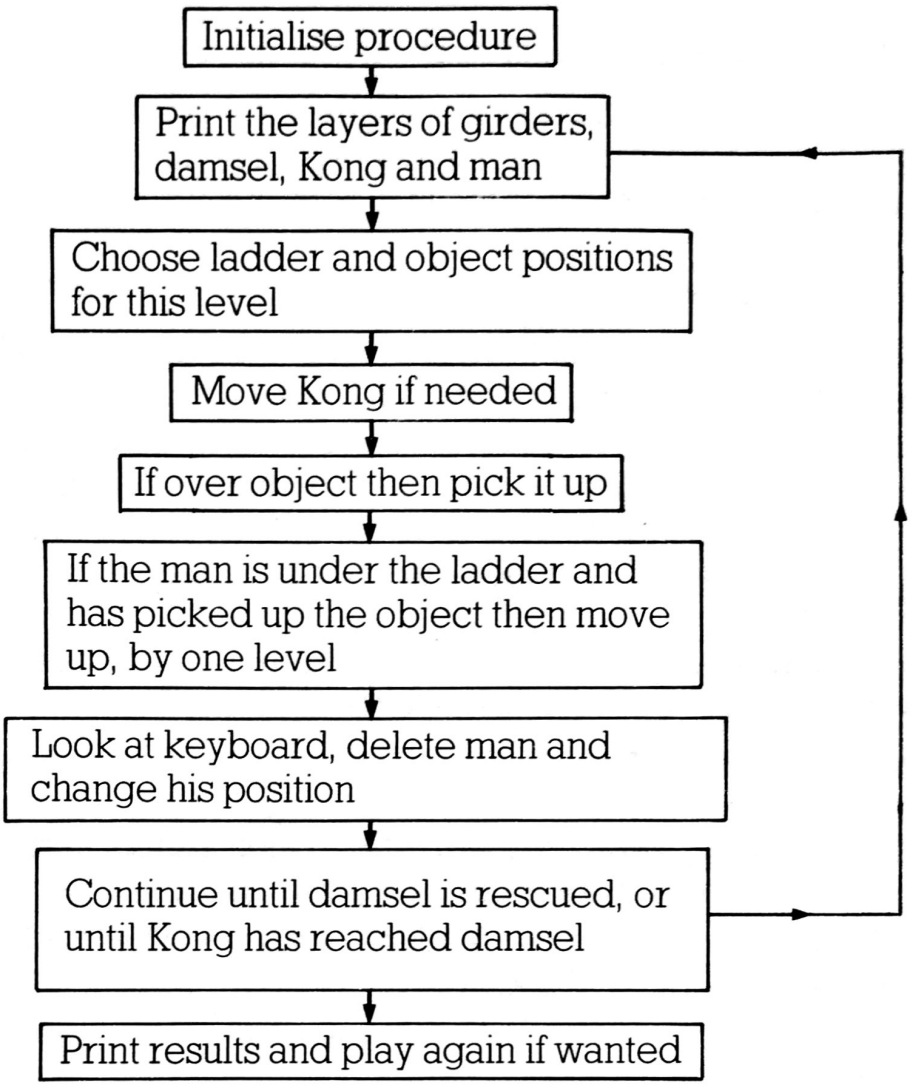
An idea for extending the program

Try a more complicated maze where the holes are remembered (in an array perhaps) and the racer can go back through them. This means dead ends could be introduced as well.

KONGO

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Define envelopes.
Define user-defined characters.
String user-defined characters together in a string variable.
Set all variables to starting values.

Print layers of girders, damsel and Kong

Print rows of user-defined girder characters in chosen colour for this level of play.
Print the damsel.
Print Kong in starting position.

Choose object and ladder positions

Pick two random numbers along girder for positions of ladder and object. If numbers match then choose again until they are different.
Print ladder and object at the two chosen points.

Decision procedures

If the counter is at a multiple of the difficulty then move Kong towards the damsel. So if the difficulty was 6, then Kong would be moved once every six counts of the counter (six times slower than the man).
If the man's column is the same as the object's then pick it up.
If the up key is being pressed and the man is carrying the object and is over the ladder, then move the position up three rows and go to the next level.
If the man is currently over the ladder then delete him by re-printing the ladder, if not print a space to delete him.

Move the man

Negative inkey produces zero when not pressed, and – 1 when pressed, so the values returned by testing for the two keys can be used in a little sum to change the man's coordinates without an IF statement (IFs are fairly slow).

Climb ladder

Reprint ladder and change character's position, by three rows, to the next level.
If the man has reached the damsel then increase the level, score and level of difficulty.

Move gorilla

Print Kong and change the column variable to be one space to the right.
Subtract 100 from the bonus added to the score for this level because Kong is one step nearer to the damsel.
If Kong has reached the damsel then end game.

End of game

Read a line of characters from a data statement into a string variable and replace any stars found with a user-defined character to form large sized letters.
Print the score.
Make a 'got her' noise!
Ask if another game is wanted; if it is then rerun, of not then change to MODE 6 and finish.

Notes

The following notes contain comments on some of the BASIC used in the program.

Negative INKEYs, for example c=INKEY(−67), have three advantages over the ordinary c=INKEY(5) type of statement in this program. It is faster than an ordinary INKEY (even if the time limit is set to zero). More than one key being pressed can be detected because it tests for individual keys. Also it does not look at the keyboard buffer, just at the key itself, so the response when playing is immediate.

The characters with ASCII codes 8 to 11 will move the cursor about on the screen when printed (for example PRINT CHR$8 will move the cursor left by one column). These characters can be put in a string variable along with the user-defined characters of a large shape (for example Kong) to make an easy way of using this shape. For example string$=CHR$224+CHR$8 +CHR$10+CHR$225 will produce, when string$ is printed, character 224 followed by a move left and then down to print character 225 underneath. VDU224,8,10,225 does the same thing but is not so friendly.

Since the gorilla only moves to the right, if the shape is extended by adding a column of spaces to be printed on it's left-hand side, then when the gorilla is printed one character to the right the spaces will blank out what is left of the previously printed shape. This means a blanking-out shape doesn't need to be used as the blanks are part of the gorilla shape.

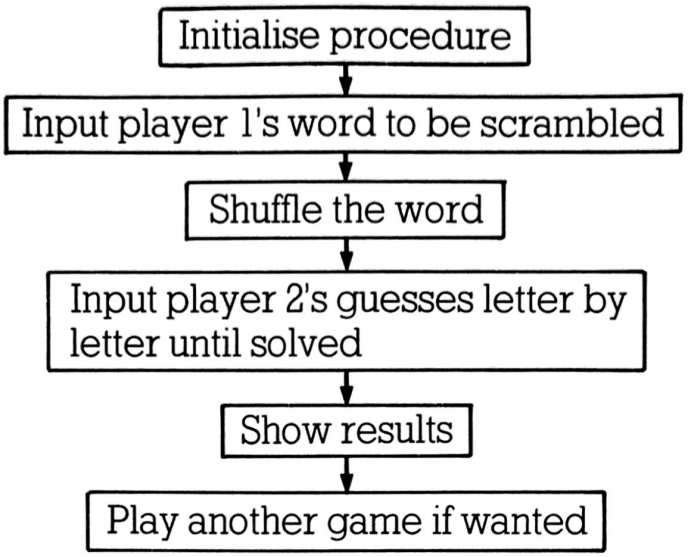
An idea for extending the program

Try animating the man. For example make two user-defined characters for each direction and print one in the even columns, one in the odd columns. Two suitably designed characters could create the impression of walking. The same kind of treatment could be applied to Kong. Try making the damsel rescue the man!

ANAGRAMS

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Set string variables to null ("") and clear screen.

Player 1 input

Get a string from player 1; if he has just pressed RETURN then repeat until he actually types a word in.
Store a copy of this word to check the guesses against.

Shuffle word

The shuffled version is built up from nothing by extracting a letter randomly from the original string and adding it to the shuffled string. The letter is then removed from the original so it isn't chosen again. Repeat this until all the letters in the original word have been used (shuffled).

Player 2 input

Print the scrambled version.
Set the error count to zero.
Using a FOR-NEXT loop let player 2 guess the word, one letter at a time.
If the guess is wrong then increase the error count by one, and make a sound.

Results

Print accolades.
Make a sound to signal success.
Print the number of errors.

Notes

The following notes contain comments on some of the BASIC used in the program.

After the random colouring of the background at the end of the game, VDU 20 is used. This resets all the colours to their default states, that is, background black, colour 1 red, and so on.
In a copy of the anagram, each time a letter is chosen it is changed to a '*' and on the screen it is coloured in. This means if there are repeated letters they won't be coloured in twice as the first occurrence has been changed to a '*'.

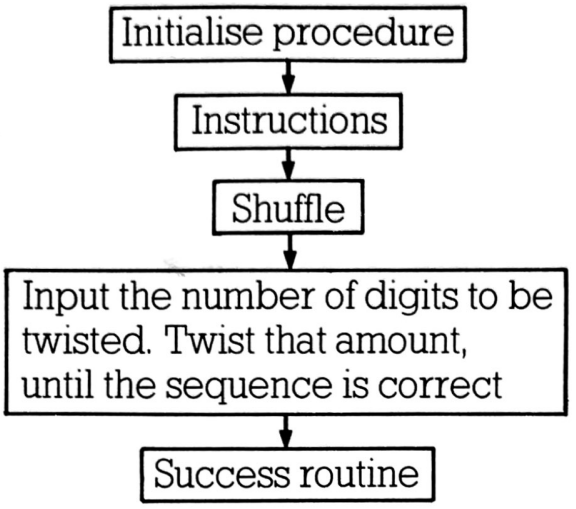
An idea for extending the program

Try adding a one player option where the computer chooses the word from a set of words in a DATA statement.

TWISTER

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail here.

Initialise procedure

Store a copy of the correct sequence and a copy to be shuffled.
Set counter to zero.
Set mode.

Instructions

Print title and explanation of the game.
Print example.
Wait for a key to be pressed before starting.

Shuffle

Similarly to the Anagrams shuffle, but the string is composed of numerical characters rather than a word typed in.
The shuffled version is created from nothing by extracting a character at random from the original and adding it to the shuffled string. The character is then removed from the original to prevent it from being chosen again. This is repeated until the original is used up (ie use up all the numerical characters in original).

Twister Routine

Check the shuffled version isn't in order already.
Store the position that the sequence is printed at, and a copy of the sequence at this stage.
Print over the previous sequence in an ordinary colour (using the stored copy of that stage and its position).
Print the new arrangement, in highlighting colour.
Decide how many characters to twist by choosing a number between 0 and 9.
Increase the tries count by one.
Take the letters in reverse order one by one from the section to be twisted, and put them (now backwards) in another string variable.
Add the right hand section to the twisted part.
Compare this version with the original to see if it is in order, repeat this routine again if it is not.

Success routine

Print the correct sequence.
Print the number of tries it took.
Flash the background and make a success sound.
Ask if another game is wanted, go back to the beginning if it is.

Notes

The following notes contain comments on some of the BASIC used in the program.

The old arrangement and position of the sequence are stored away so that it can be printed over later in the non-highlighted way when the new arrangement is printed up in highlights.

POS is a BASIC function to give the current column of the text cursor and VPOS gives the current row.

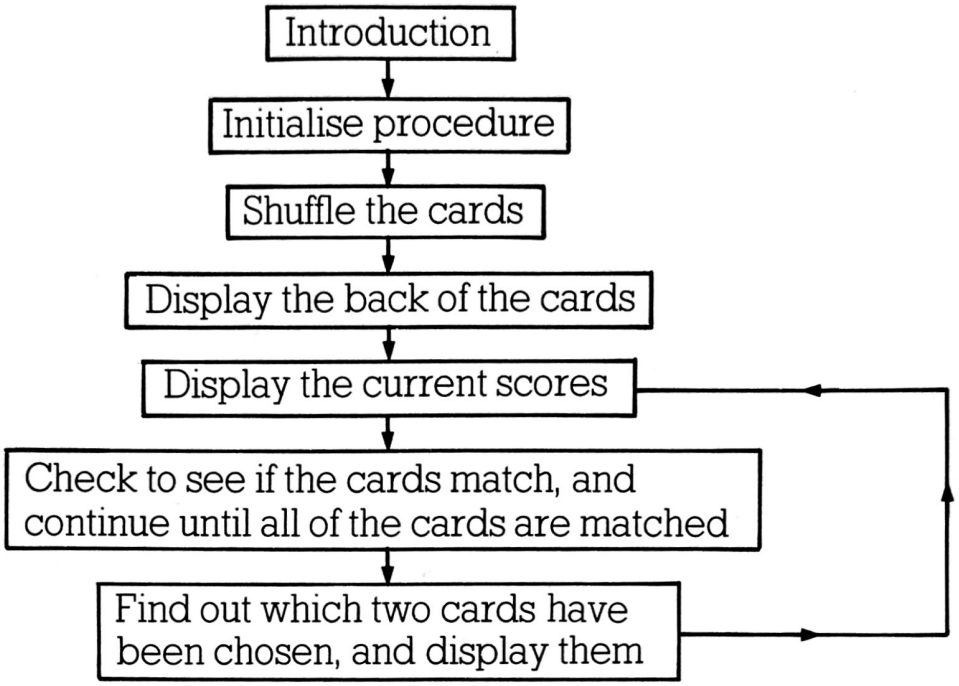
An idea for extending the program

Use a longer sentence, and enable the player to twist any part of that sequence eg from position 5 to position 12.

MATCHEM

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail.

Introduction

Print a welcoming screen.
Find out whether one or two players are taking part. If there are two players then ask their names and set up for two players.

Initialise procedure

Zero the scores and number of guesses taken.
Set up variables, particularly a string variable to represent the cards.

Shuffle cards

This is the same theory as Anagrams and Twister, but the characters in the string now represent the 20 cards.

Display the back of the cards

Use two loops: one to go along the columns, one to go down the rows.
Print the back (ie the number from 1 to 20 which you need to type to turn that card over) at the relevant position.

Display scores

If there is only one player then print his score.
If there are two players then carry out the procedure to swap turns, print both scores, and print whose turn it is along with effects.

Choose cards

Get the number of the card chosen.
Find which card it is, and if it has already been matched, or the number is too large or small, then ask for another choice.
Work out the column and row of this card.
Look through the list of cards until the chosen one is found, then print the shapes for that card at the column and row worked out for it.
Choose a second card in the same way, ensuring it is different from the first, and show the card in the same way as well.

Check for match

Increase guess count by one.
If the shape on the first card matches the shape on the second card then mark those two as matched, increase the score of the player, and loop back for the next go.
If the cards don't match then make a sound, and turn the cards over again so the number is showing.
Loop round for another go until all the cards have been matched up.
When all the cards are matched print a finish message and ask if another game is wanted.

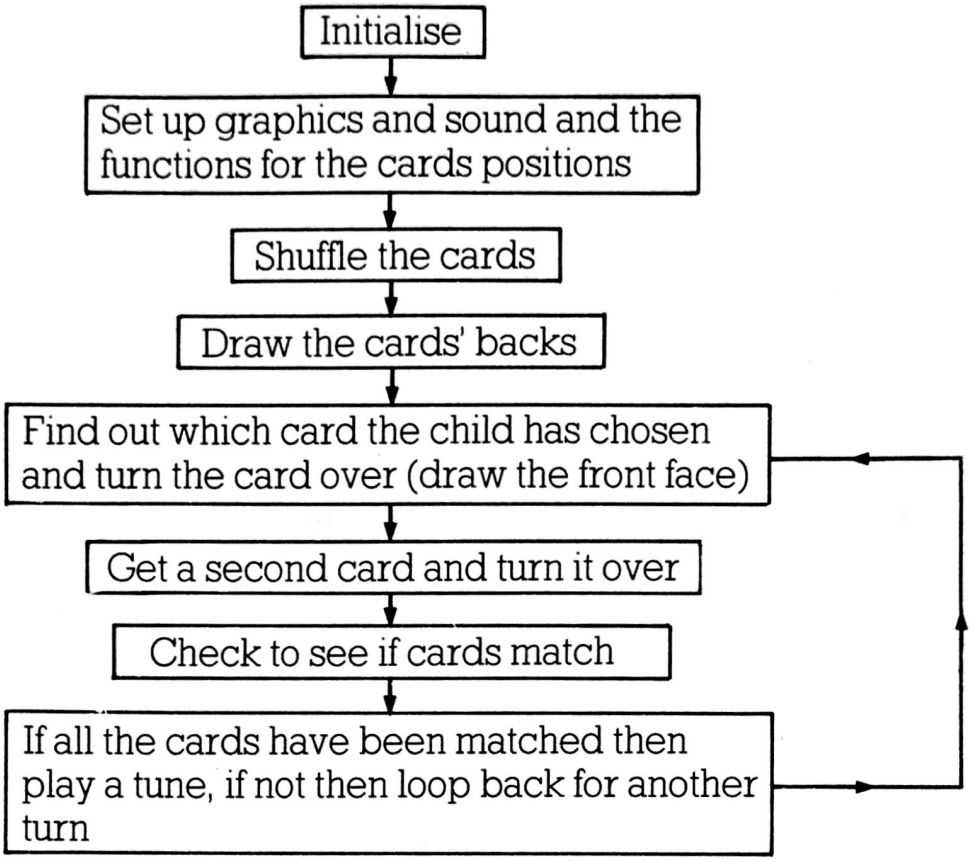
An idea for extending the program

Change the characters to user-defined characters. Alter the game to allow one player to continue his go until he makes an incorrect match.

PAIRS

Program outline

Below is an outline of the program.



Program structure

Each procedure making up the program outline is broken down into more detail.

Initialise

Set all variables to starting values.
Dimension array.
Define any envelopes needed.
Choose a tune at random.
Define functions to give the column and row of each card from its number.

Shuffle cards

This is the same shuffle routine as explained for Anagrams, Twister and Pelmanism, but for even fewer characters (6).

Draw cards' backs

Print each card's number at the relevant column and row.
Draw a frame round each card.

Get which card has been chosen and turn it over

Get the number of a card.
Check it has not been chosen, or isn't too large or small a number.
Define the characters for the shape of this card, and its colours.
Print this card at the relevant coordinates.

Get a second card and turn it over

This is done in the same way as the first card, but check that they are different cards.
Make a different sound for each type of card turned over.

Check for a match

If the type of each card is the same then make a success sound and increase the score by one.
If the cards do not match then make an unsuccessful sound and turn them both over again (just showing their numbers).

Check if all the cards are matched

If the score is 3 then all the cards have been matched, so play the tune selected randomly from the three available, and flash the background.
Then ask if another game is wanted.
If all the cards are not matched then loop back to get another two choices.

Notes

The following notes contain comments on some of the BASIC used in the program. Each time a card is chosen, its picture is found and the 16 characters used each time for the pictures are redefined to the correct picture, (by RESTOREing to the right lines of data) and printed out.

Since there are four types of picture, one has to be left out of each game as there are only three pairs. This is done just before the shuffle routine.

DEF FN is used similarly to DEF PROC here. It is more convenient than a procedure for these two one-line functions.

An idea for extending the program

Try showing all the cards before the game is started. This gives the player a chance of remembering them (this may possibly require a few more cards to make it hard enough).

ACORNSOFT

SLX18/B